Tutorial 03: KNN and Decision Tree Implementation

The Following tutorial contains required data pre-processing , implementation of KNN and Decision Tree Classifier

You are required to create a jupyter notebook that contains your results of undertaking the research detailed in the attached file.

Topics covered.

1. Data pre-processing
2. Data visualization
3. Train and test data split
4. Implementation of KNN and Decision Tree
5. Prediction and accuracy validation

Note: The notebook(.ipynb or .html) can be downloaded from GitHub.

To execute the code, click on the corresponding cell and press the SHIFT-ENTER keys simultaneously.

Original notebook by Ninja Programming Corp YouTube, 2021.

Stay Connected

3.0.0 Required library description

- Numpy

Numpy, which stands for numerical Python, is a Python library package to support numerical computations.Numpy provides a suite of functions that can efficiently manipulate elements of the ndarray.

- MatplotLib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.Matplotlib produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

- pandas

pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

- Sklearn

It is a Python library that offers various features for data processing that can be used for classification, clustering, and model selection.

Installing required libraries

In [1]:
```
!pip install pandas
!pip install numpy
!pip install matplotlib
!pip install sklearn
```

3.1.0 Iris Dataset

The Iris Dataset contains four features (length and width of sepals and petals) of 50 samples of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). These measures were used to create a linear discriminant model to classify the species. The dataset is often used in data mining, classification and clustering examples and to test algorithms.

3.1.1. Import Library

Following are the necessary library we will be using

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

3.1.2 Import dataset

In [3]:
```
from sklearn.datasets import load_iris
```

3.1.3 Loading the Dataset

In [4]:
```
iris = load_iris()
```

In [5]:
```
iris
```

Out[5]:
```
{'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
        [5.1, 3.8, 1.5, 0.3],
        [5.4, 3.4, 1.7, 0.2],
        [5.1, 3.7, 1.5, 0.4],
```

[5.1, 3.7, 1.5, 0.4],
[4.6, 3.6, 1. , 0.2],
[5.1, 3.3, 1.7, 0.5],
[4.8, 3.4, 1.9, 0.2],
[5. , 3. , 1.6, 0.2],
[5. , 3.4, 1.6, 0.4],
[5.2, 3.5, 1.5, 0.2],
[5.2, 3.4, 1.4, 0.2],
[4.7, 3.2, 1.6, 0.2],
[4.8, 3.1, 1.6, 0.2],
[5.4, 3.4, 1.5, 0.4],
[5.2, 4.1, 1.5, 0.1],
[5.5, 4.2, 1.4, 0.2],
[4.9, 3.1, 1.5, 0.2],
[5. , 3.2, 1.2, 0.2],
[5.5, 3.5, 1.3, 0.2],
[4.9, 3.6, 1.4, 0.1],
[4.4, 3. , 1.3, 0.2],
[5.1, 3.4, 1.5, 0.2],
[5. , 3.5, 1.3, 0.3],
[4.5, 2.3, 1.3, 0.3],
[4.4, 3.2, 1.3, 0.2],
[5. , 3.5, 1.6, 0.6],
[5.1, 3.8, 1.9, 0.4],
[4.8, 3. , 1.4, 0.3],
[5.1, 3.8, 1.6, 0.2],
[4.6, 3.2, 1.4, 0.2],
[5.3, 3.7, 1.5, 0.2],
[5. , 3.3, 1.4, 0.2],
[7. , 3.2, 4.7, 1.4],
[6.4, 3.2, 4.5, 1.5],
[6.9, 3.1, 4.9, 1.5],
[5.5, 2.3, 4. , 1.3],
[6.5, 2.8, 4.6, 1.5],
[5.7, 2.8, 4.5, 1.3],
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1.],
[6.6, 2.9, 4.6, 1.3],
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1.],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1.],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1.],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1.],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1.],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1.],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2.],
[6.4, 2.7, 5.3, 1.9],
[6.8, 3. , 5.5, 2.1],
[5.7, 2.5, 5. , 2.],
[5.8, 2.8, 5.1, 2.4],
[6.4, 3.2, 5.3, 2.3],
[6.5, 3. , 5.5, 1.8],
[7.7, 3.8, 6.7, 2.2],
[7.7, 2.6, 6.9, 2.3],
[6. , 2.2, 5. , 1.5],
[6.9, 3.2, 5.7, 2.3],
[5.6, 2.8, 4.9, 2.],
[7.7, 2.8, 6.7, 2.],
[6.3, 2.7, 4.9, 1.8],
[6.7, 3.3, 5.7, 2.1],
[7.2, 3.2, 6. , 1.8],
[6.2, 2.8, 4.8, 1.8],
[6.1, 3. , 4.9, 1.8],
[6.4, 2.8, 5.6, 2.1],
[7.2, 3. , 5.8, 1.6],
[7.4, 2.8, 6.1, 1.9],
[7.9, 3.8, 6.4, 2.],
[6.4, 2.8, 5.6, 2.2],
[6.3, 2.8, 5.1, 1.5],
[6.1, 2.6, 5.6, 1.4],
[7.7, 3. , 6.1, 2.3],
[6.3, 3.4, 5.6, 2.4],
[6.4, 3.1, 5.5, 1.8],
[6. , 3. , 4.8, 1.8],
[6.9, 3.1, 5.4, 2.1],
[6.7, 3.1, 5.6, 2.4],
[6.9, 3.1, 5.1, 2.3],
[5.8, 2.7, 5.1, 1.9],
[6.8, 3.2, 5.9, 2.3],
[6.7, 3.3, 5.7, 2.5],
[6.7, 3. , 5.2, 2.3],
[6.3, 2.5, 5. , 1.9],
[6.5, 3. , 5.2, 2.],

```
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
 'frame': None,
 'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
 'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data Set Characteristics:**\n\n    :Number of Instances: 150 (50 in each of three classes)\n    :Number of Attributes: 4 numeric, predictive attributes and the class\n    :Attribute Information:\n        - sepal length in cm\n        - sepal width in cm\n        - petal length in cm\n        - petal width in cm\n        - class:\n                - Iris-Setosa\n                - Iris-Versicolour\n                - Iris-Virginica\n                \n    :Summary Statistics:\n\n    ============== ==== ==== ======= ===== ====================\n                    Min  Max   Mean    SD   Class Correlation\n    ============== ==== ==== ======= ===== ====================\n    sepal length:   4.3  7.9   5.84   0.83    0.7826\n    sepal width:    2.0  4.4   3.05   0.43   -0.4194\n    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)\n    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)\n    ============== ==== ==== ======= ===== ====================\n\n    :Missing Attribute Values: None\n    :Class Distribution: 33.3% for each of 3 classes.\n    :Creator: R.A. Fisher\n    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n    :Date: July, 1988\n\nThe famous Iris database, first used by Sir R.A. Fisher. The dataset is taken\nfrom Fisher\'s paper. Note that it\'s the same as in R, but not as in the UCI\nMachine Learning Repository, which has two wrong data points.\n\nThis is perhaps the best known database to be found in the\npattern recognition literature.  Fisher\'s paper is a classic in the field and\nis referenced frequently to this day.  (See Duda & Hart, for example.)  The\ndata set contains 3 classes of 50 instances each, where each class refers to a\ntype of iris plant.  One class is linearly separable from the other 2; the\nlatter are NOT linearly separable from each other.\n\n.. topic:: References\n\n   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"\n     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to\n     Mathematical Statistics" (John Wiley, NY, 1950).\n   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.\n     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.\n   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System\n     Structure and Classification Rule for Recognition in Partially Exposed\n     Environments".  IEEE Transactions on Pattern Analysis and Machine\n     Intelligence, Vol. PAMI-2, No. 1, 67-71.\n   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions\n     on Information Theory, May 1972, 431-433.\n   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II\n     conceptual clustering system finds 3 classes in the data.\n   - Many, many more ...',
 'feature_names': ['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': 'c:\\users\\bren saud\\appdata\\local\\programs\\python\\python39\\lib\\site-packages\\sklearn\\datasets\\data\\iris.csv'}
```

3.1.4 Checking the shape of the database

In [6]:
```
iris.data.shape
```

Out[6]:
```
(150, 4)
```

In [7]:
```
data_iris = iris.data
```

In [8]:
```
data_iris
```

Out[8]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
       [6.3, 3.3, 4.7, 1.6],
       [4.9, 2.4, 3.3, 1. ],
       [6.6, 2.9, 4.6, 1.3],
       [5.2, 2.7, 3.9, 1.4],
       [5. , 2. , 3.5, 1. ],
       [5.9, 3. , 4.2, 1.5],
       [6. , 2.2, 4. , 1. ],
       [6.1, 2.9, 4.7, 1.4],
       [5.6, 2.9, 3.6, 1.3],
       [6.7, 3.1, 4.4, 1.4],
       [5.6, 3. , 4.5, 1.5],
       [5.8, 2.7, 4.1, 1. ],
       [6.2, 2.2, 4.5, 1.5],
       [5.6, 2.5, 3.9, 1.1],
       [5.9, 3.2, 4.8, 1.8],
       [6.1, 2.8, 4. , 1.3],
       [6.3, 2.5, 4.9, 1.5],
       [6.1, 2.8, 4.7, 1.2],
       [6.4, 2.9, 4.3, 1.3],
       [6.6, 3. , 4.4, 1.4],
```

```
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
[6.4, 2.7, 5.3, 1.9],
[6.8, 3. , 5.5, 2.1],
[5.7, 2.5, 5. , 2. ],
[5.8, 2.8, 5.1, 2.4],
[6.4, 3.2, 5.3, 2.3],
[6.5, 3. , 5.5, 1.8],
[7.7, 3.8, 6.7, 2.2],
[7.7, 2.6, 6.9, 2.3],
[6. , 2.2, 5. , 1.5],
[6.9, 3.2, 5.7, 2.3],
[5.6, 2.8, 4.9, 2. ],
[7.7, 2.8, 6.7, 2. ],
[6.3, 2.7, 4.9, 1.8],
[6.7, 3.3, 5.7, 2.1],
[7.2, 3.2, 6. , 1.8],
[6.2, 2.8, 4.8, 1.8],
[6.1, 3. , 4.9, 1.8],
[6.4, 2.8, 5.6, 2.1],
[7.2, 3. , 5.8, 1.6],
[7.4, 2.8, 6.1, 1.9],
[7.9, 3.8, 6.4, 2. ],
[6.4, 2.8, 5.6, 2.2],
[6.3, 2.8, 5.1, 1.5],
[6.1, 2.6, 5.6, 1.4],
[7.7, 3. , 6.1, 2.3],
[6.3, 3.4, 5.6, 2.4],
[6.4, 3.1, 5.5, 1.8],
[6. , 3. , 4.8, 1.8],
[6.9, 3.1, 5.4, 2.1],
[6.7, 3.1, 5.6, 2.4],
[6.9, 3.1, 5.1, 2.3],
[5.8, 2.7, 5.1, 1.9],
[6.8, 3.2, 5.9, 2.3],
[6.7, 3.3, 5.7, 2.5],
[6.7, 3. , 5.2, 2.3],
[6.3, 2.5, 5. , 1.9],
[6.5, 3. , 5.2, 2. ],
[6.2, 3.4, 5.4, 2.3],
[5.9, 3. , 5.1, 1.8]])
```

3.1.5 Converting the data into a dataframe and giving proper column names

As we know we can give proper column name while creating dataframe so we have made a list matching the row length and passed in columns parameter.

In [9]:
```
iris_features = pd.DataFrame(data_iris, columns= ['Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Width'])
```

In [10]:
```
iris_features
```

Out[10]:

	Sepal Length	Sepal Width	Petal Length	Petal Width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

3.1.6 accessing the target values from dataset using attribute

In [11]:
```
iris_target = iris.target
```

3.1.7 making a seprate dataframe for target values with proper column name

here, after getting the target values we will now create a dataframe using target values with proper column name

In [12]:
```
iris_target = pd.DataFrame(iris_target, columns = ['Target'] )
```

3.1.8 Checking the shape

The target value is a one dimensional array. At first it had 150 data in single array but now it has 150 rows and single column. This is because we have converted it in to dataframe so it becomes two dimensional.

In [13]:
```
iris_target.shape
```

Out[13]:
```
(150, 1)
```

In [14]:
```
iris_target
```

Out[14]:

	Target
0	0
1	0
2	0
3	0
4	0
...	...
145	2
146	2
147	2
148	2
149	2

150 rows × 1 columns

3.1.9 Concatination

Now we can concat those taget values to our iris_feature values. Here we have passed the dataframes which we want to concat in a list and given axis as 1 or column.

Giving axis as 1 is important because we want to concat them in x asis or horizontal axis.

In [15]:
```
data_iris = pd.concat([iris_features, iris_target], axis = 1)
```

In [16]:
```
data_iris.head()
```

Out[16]:

	Sepal Length	Sepal Width	Petal Length	Petal Width	Target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

3.1.10 Checking some random middle values in the dataframe

In [17]:
```
data_iris.loc[5:6]
```

Out[17]:

	Sepal Length	Sepal Width	Petal Length	Petal Width	Target
5	5.4	3.9	1.7	0.4	0
6	4.6	3.4	1.4	0.3	0

3.1.11 selecting only two columns

In [18]:
```
data_iris.iloc[:,:2]
```

Out[18]:

	Sepal Length	Sepal Width
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2
3	4.6	3.1
4	5.0	3.6
...	...	...
145	6.7	3.0
146	6.3	2.5
147	6.5	3.0
148	6.2	3.4
149	5.9	3.0

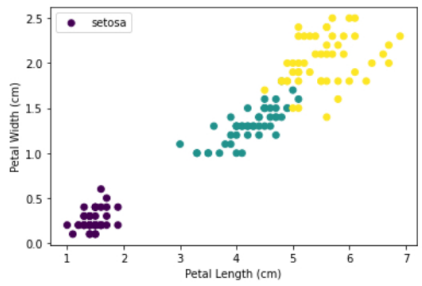
150 rows × 2 columns

3.2.1 Visualizing the dataset

here we are visualizing two values and there relationship. we have passed values of petal lenght in x asis, and petal width in y axis.

C parameter is used for passing different values to get different colors. so we are passing target values

In [19]:
```
plt.scatter(data_iris.iloc[:,2], data_iris.iloc[:,3], c = iris.target)
plt.xlabel('Petal Length (cm)')
plt.ylabel('Petal Width (cm)')
plt.legend(iris.target_names)
plt.show()
```



3.2.2 Ploting data by differentiating Target values

Here we are ploting each data for two class but now storing the values differently and applying different color for different target values.

- for target in np.unique(iris.target):

  Selects unique values and make a list out of that values
- x = [iris.data[i,2] for i in range(len(iris.target)) if iris.target[i]==target]

  select all the values from column in 2nd index (.i.e. Petal Length) where column 'target' values matches the condition
- y = [iris.data[i,2] for i in range(len(iris.target)) if iris.target[i]==target]

  select all the values from column in 3rd index (.i.e. Petal Width) where column 'target' values matches the
- plt.scatter(x, y, color=['red', 'blue', 'green'][target], label=iris.target_names[target])

  first we pass the item we want to plot and then we are passing color values for our 3 different targets and assigning names for that values.

We then give name to labels, provide proper title legend is used to provide discription about the plots and lebels

Note: The values of x and y are ploted 3 times in same plot because of loop for 3 target classes

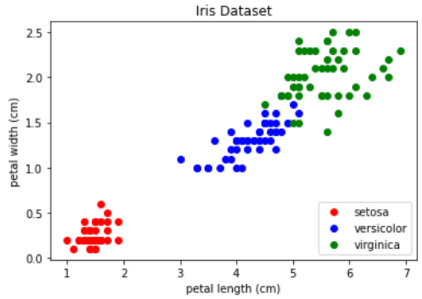
In [20]:
```
for target in  np.unique(iris.target):
    print(target)
    x = [iris.data[i,2] for i in range(len(iris.target)) if iris.target[i]==target]


    y = [iris.data[i,3] for i in range(len(iris.target)) if iris.target[i]==target]

    plt.scatter(x, y, c=['red', 'blue', 'green'][target], label=iris.target_names[target])


plt.xlabel(iris.feature_names[2])
plt.ylabel(iris.feature_names[3])
plt.title('Iris Dataset')
plt.legend(iris.target_names, loc='lower right')
plt.show()
```

```
0
1
2
```



3.2.3 Spliting independent features or X and Y class data

as we know our independent features are upto 4th column index of dataframe we select them and store in a variable

In [21]:
```
x_data = data_iris.iloc[:,:4]
```

In [22]:
```
x_data
```

Out[22]:

	Sepal Length	Sepal Width	Petal Length	Petal Width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0

148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

3.2.4 Spliting dependent features or Y data. Data to be predicted

as we know our dependent feature is in our last column index. so we select our last column value

In [23]:
```
y_data = data_iris.iloc[:,-1]
```

In [24]:
```
y_data
```

Out[24]:
```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: Target, Length: 150, dtype: int32
```

3.3.0 K-Nearest Neighbor (KNN)

K-nearest neighbors (KNN) algorithm uses 'feature similarity' to predict the values of new datapoints which further means that the new data point will be assigned a value based on how closely it matches the points in the training set. We can understand its working with the help of following steps –

3.3.1 Steps of KNN

- Step 1 – For implementing any algorithm, we need dataset. So during the first step of KNN, we must load the training as well as test data.
- Step 2 – Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.
- Step 3 – For each point in the test data do the following –
  - 3.1 – Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.
  - 3.2 – Now, based on the distance value, sort them in ascending order.
  - 3.3 – Next, it will choose the top K rows from the sorted array.
  - 3.4 – Now, it will assign a class to the test point based on most frequent class of these rows.
- Step 4 – End

3.3.2 KNN Implementation

First we will import KNeighborsClassifier from sklearn.neighbors pacakages

Then we instanciate the algorithm and pass k value. After that p value will be given to specify which distance algrothim to use.

```
When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance
(l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.
```

after that we train with fit function passing the Independent features and dependent features .i.e. X_data and Y_data

In [25]:
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors = 6, metric = 'minkowski', p = 1)
knn.fit(x_data,  y_data)
```

Out[25]:
```
KNeighborsClassifier(n_neighbors=6, p=1)
```

3.3.4 Making new data to make prediction

In [26]:
```
X_new  = np.array([[5.6, 3.4, 1.4, 0.1]])
```

In [27]:
```
X_new
```

Out[27]:
```
array([[5.6, 3.4, 1.4, 0.1]])
```

3.3.5 Passing those values in the model and running predict function

here, different values represent different classes.

0 = setosa
1 = versicolor
2 = verginica

In [28]:
```
knn.predict(X_new)
```

Out[28]:
```
array([0])
```

In [29]:
```
X_new_2 = np.array([[7.5,4, 5.2, 2]])
```

In [30]:
```
knn.predict(X_new_2)
```

Out[30]:
```
array([2])
```

3.4.0 Train Test Spilt

train_test_split is a function in Sklearn model selection for splitting data arrays into two subsets: for training data and for testing data. With this function, you don't need to divide the dataset manually.

By default, Sklearn train_test_split will make random partitions for the two subsets. However, you can also specify a random state for the operation.

In [31]:
```
from sklearn.model_selection import train_test_split
```

3.4.1 Data Split

Spliting the data into Train, Test for X(independent features) and Y (Dependent features or label to be predected)

First we pass the X_data and Y_data then give trian size which will determine how many percentage of the data to be used as traindata and test data

In [32]:
```
X_train,X_test,Y_train,Y_test = train_test_split(x_data,y_data,train_size=0.70,random_state=1)
```

3.4.2 Shape of X Train data

In [33]:
```
X_train.shape
```

Out[33]:
```
(105, 4)
```

3.4.3 Shape of X_Test data

In [34]:
```
X_test.shape
```

Out[34]:
```
(45, 4)
```

3.4.4 Shape of Y Train data

In [35]:
```
Y_train.shape
```

Out[35]:
```
(105,)
```

3.4.5 Shape of Y Test data

In [36]:
```
Y_test.shape
```

Out[36]:
```
(45,)
```

3.4.6 Train model

Training the KNN model with with train data

In [37]:
```
new_knn_model = knn.fit(X_train, Y_train)
```

3.4.7 Printing the model details

In [38]:
```
new_knn_model
```

Out[38]:
```
KNeighborsClassifier(n_neighbors=6, p=1)
```

3.4.8 Pridicting with X_test data

In [39]:
```
predicted_types = new_knn_model.predict(X_test)
```

3.4.9 Importing metrics for Testing

In [40]:
```
from sklearn import metrics
```

3.4.10 Testing the accuracy of the model

In [41]:
```
metrics.accuracy_score(Y_test, predicted_types)
```

Out[41]:
```
0.9777777777777777
```

3.5.0 Decision Tree Classifier

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches

- Root Node: Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
- Leaf Node: Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- Splitting: Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- Branch/Sub Tree: A tree formed by splitting the tree.
- Pruning: Pruning is the process of removing the unwanted branches from the tree.
- Parent/Child node: The root node of the tree is called the parent node, and other nodes are called the child nodes.

3.5.1 Steps in Decision tree

- Step-1: Begin the tree with the root node, says S, which contains the complete dataset.
- Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).
- Step-3: Divide the S into subsets that contains possible values for the best attributes.
- Step-4: Generate the decision tree node, which contains the best attribute.
- Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called

the final node as a leaf node.

3.5.2 Importing necessary pacakage from sklearn

In [42]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

3.5.3 Model instantiate

In [43]:
```
model_dt = DecisionTreeClassifier()
```

3.5.2 Training with our training data

In [44]:
```
model_dt.fit(X_train, Y_train)
```

Out[44]:
```
DecisionTreeClassifier()
```

3.5.3 Pridicting the values in our Test data

In [45]:
```
predict_dt = model_dt.predict(X_test)
```

3.5.4 Testing the accuracy of the model

In [46]:
```
accuracy_score(Y_test, predict_dt)
```

Out[46]:
```
0.9555555555555556
```

3.5.5 Importing cross_validation_score pacakage for model validation

In [47]:
```
from sklearn.model_selection import cross_val_score
```

3.5.6 Passing the model which we want to validate

In [48]:
```
scores_DT = cross_val_score(model_dt, X_test, Y_test, cv = 10)
```

In [49]:
```
scores_DT
```

Out[49]:
```
array([1.  , 1.  , 1.  , 0.8 , 1.  , 1.  , 0.75, 1.  , 0.75, 1.  ])
```

😊 Thanks for your time 😊

What do you think of this **"Decision Tree and K Nearest Neighbor classification"?**(Appreciation, Suggestions, and Questions are highly appreciated).